CHAPTER 12

Managing Disks and Filesystems

IN THIS CHAPTER

Working with shell scripts

Creating logical volumes with LVM

Adding filesystems

Mounting filesystems

Unmounting filesystems

Your operating system, applications, and data all need to be kept on some kind of permanent storage so that when you turn your computer off and then on again, it is all still there. Traditionally, that storage has been provided by a hard disk in your computer. To organize the information on that disk, the disk is usually divided into partitions, with most partitions given a structure referred to as a *filesystem*.

This chapter describes how to work with hard drives. Hard drive tasks include partitioning, adding filesystems, and managing those filesystems in various ways. Storage devices that are attached to the systems such as removable devices, including hard disk drives (HDDs) and solid-state drives (SSDs), and network devices can be partitioned and managed in the same ways.

After covering basic partitions, I describe how Logical Volume Manager (LVM) can be used to make it easier to grow, shrink, and otherwise manage filesystems more efficiently.

Understanding Disk Storage

The basics of how data storage works are the same in most modern operating systems. When you install the operating system, the disk is divided into one or more partitions. Each partition is formatted with a filesystem. In the case of Linux, some of the partitions may be specially formatted for elements such as swap area or LVM physical volumes. Disks are used for permanent storage; *random access memory (RAM)* and swap are used for temporary storage. For example, when you run a command, that command is copied from the hard disk into RAM so that your computer processor (CPU) can access it more quickly.

Linux® Bible, Tenth Edition. Christopher Negus.

Your CPU can access data much faster from RAM than it can from a hard disk, although SSDs are more like RAM than HDDs. However, a disk is usually much larger than RAM, RAM is much more expensive, and RAM is erased when the computer reboots. Think of your office as a metaphor for RAM and disk. A disk is like a file cabinet where you store folders of information you need. RAM is like the top of your desk, where you put the folder of papers while you are using it but put it back in the file cabinet when you are not.

If RAM fills up by running too many processes or a process with a memory leak, new processes fail if your system doesn't have a way to extend system memory. That's where a swap area comes in. A *swap space* is a hard disk swap partition or a swap file where your computer can "swap out" data from RAM that isn't being used at the moment and then "swap in" the data back to RAM when it is again needed. Although it is better never to exceed your RAM (performance takes a hit when you swap), swapping out is better than having processes just fail.

Another special partition is a *Logical Volume Manager (LVM)* physical volume. LVM physical volumes enable you to create pools of storage space called *volume groups*. From those volume groups, you have much more flexibility for growing and shrinking logical volumes than you have resizing disk partitions directly.

For Linux, at least one disk partition is required, assigned to the root `(/)` of the entire Linux filesystem. However, it is more common to have separate partitions that are assigned to particular directories, such as `/home`, `/var`, and/or `/tmp`. Each of the partitions is connected to the larger Linux filesystem by mounting it to a point in the filesystem where you want that partition to be used. Any file added to the mount point directory of a partition, or a subdirectory, is stored on that partition.

NOTE

The word *mount* refers to the action of connecting a filesystem from a hard disk, USB drive, or network storage device to a particular point in the filesystem. This action is done using the `mount` command, along with options to tell the command where the storage device is located and to which directory in the filesystem to connect it.

The business of connecting disk partitions to the Linux filesystem is done automatically and invisibly to the end user. How does this happen? Each regular disk partition created when you install Linux is associated with a device name. An entry in the `/etc/fstab` file tells Linux each partition's device name and where to mount it (as well as other bits of information). The mounting is done when the system boots.

Most of this chapter focuses on understanding how your computer's disk is partitioned and connected to form your Linux filesystem as well as how to partition disks, format filesystems and swap space, and have those items used when the system boots. The chapter then covers how to do partitioning and filesystem creation manually.

Coming from Windows

Filesystems are organized differently in Linux than they are in Microsoft Windows operating systems. Instead of drive letters (for example, `A:`, `B:`, `C:`) for each local disk, network filesystem, CD-ROM, or other type of storage medium, everything fits neatly into the Linux directory structure.

Some drives are connected (mounted) automatically into the filesystem when you insert removable media. For example, a CD might be mounted on `/media/cdrom`. If the drive isn't mounted automatically, it is up to an administrator to create a mount point in the filesystem and then connect the disk to that point.

Linux can understand VFAT filesystems, which are often the default format when you buy a USB flash drive. A VFAT and exFAT USB flash drive provides a good way to share data between Linux and Windows systems. Linux kernel support is available for NTFS filesystems, which are usually used with Windows these days. However, NTFS, and sometimes exFAT, often require that you install additional kernel drivers in Linux.

VFAT file systems are often used when files need to be exchanged between different types of operating systems. Because VFAT was used in MS-DOS and early Windows operating systems, it offers a good lowest common denominator for sharing files with many types of systems (including Linux). NTFS is the file system type most commonly used with modern Microsoft Windows systems.

Partitioning Hard Disks

Linux provides several tools for managing your hard disk partitions. You need to know how to partition your disk if you want to add a disk to your system or change your existing disk configuration.

The following sections demonstrate disk partitioning using a removable USB flash drive and a fixed hard disk. To be safe, I use a USB flash drive that doesn't contain any data that I want to keep in order to practice partitioning.

Changing partitioning can make a system unbootable!

I don't recommend using your system's primary hard disk to practice changing partitioning because a mistake can make your system unbootable. Even if you use a separate USB flash drive to practice, a bad entry in `/etc/fstab` can hang your system on reboot. If after changing partitions your system fails to boot, refer to Chapter 21, "Troubleshooting Linux," for information on how to fix the problem.

Understanding partition tables

PC architecture computers have traditionally used *Master Boot Record (MBR) partition tables* to store information about the sizes and layouts of the hard disk partitions. There are

many tools for managing MBR partitions that are quite stable and well known. A few years ago, however, a new standard called *Globally Unique Identifier (GUID) partition tables* began being used on systems as part of the UEFI computer architecture to replace the older BIOS method of booting the system.

Many Linux partitioning tools have been updated to handle GUID partition tables (gpt). Other tools for handling GUID partition tables have been added. Because the popular `fdisk` command does not support gpt partitions, the `parted` command is used to illustrate partitioning in this chapter.

Limitations imposed by the MBR specification brought about the need for GUID partitions. In particular, MBR partitions are limited to 2TB in size. GUID partitions can create partitions up to 9.4ZB (zettabytes).

Viewing disk partitions

To view disk partitions, use the `parted` command with the `-l` option. The following is an example of partitioning on a 160GB fixed hard drive on a Red Hat Enterprise Linux 8 system:

```
# parted -l /dev/sda
Disk /dev/sda: 160.0 GB, 160000000000 bytes, 312500000 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk label type: dos
Disk identifier: 0x0008870c
   Device Boot      Start         End      Blocks   Id  System
/dev/sda1   *        2048     1026047      512000   83  Linux
/dev/sda2         1026048   304281599   151627776   8e  Linux LVM
```

When a USB flash drive is inserted, it is assigned to the next available `sd` device. The following example shows the partitioning on the hard drive (`/dev/sda`) and a USB drive from a Fedora 30 system, where `/dev/sdb` is assigned as the USB device name (the second disk on the system). This USB drive is a new 128GB USB flash drive:

```
# fdisk -l /dev/sdb
```

Although this drive was assigned to`/dev/sdb`, your drive might be assigned to a different device name. Here are some things to look for:

- A SCSI or USB storage device, represented by an *sd?* device (such as `sda`, `sdb`, `sdc`, and so on) can have up to 16 minor devices (for example, the main `/dev/sdc` device and `/dev/sdc1` through `/dev/sdc15`). So, there can be 15 partitions total. A NVMe SSD storage device, represented by a `nvme` device (such as `nvme0`, `nvme1`, `nvme2`, and so on) can be divided into one or more namespaces (most devices just use the first namespace) and partitions. For example, `/dev/nvme0n1p1` represents the first partition in the first namespace on the first NVMe SSD.
- For x86 computers, disks can have up to four primary partitions. So, to have more than four total partitions, one must be an extended partition. Any partitions

beyond the four primary partitions are logical partitions that use space from the extended partition.

- The `id` field indicates the type of partition. Notice that there is a Linux LVM partition in the first example.

Your first primary hard disk usually appears as `/dev/sda`. With RHEL and Fedora installations, there is usually at least one LVM partition created by the installer, out of which other partitions can be assigned. So, the output of `fdisk` might be as simple as the following:

```
# parted -l
Disk /dev/sda: 500.1 GB, 500107862016 bytes
```

The first partition is roughly 210MB and is mounted on the `/boot/efi` directory. The second partition (1074MB) is mounted on the `/boot` partition. For older MBR partition tables, there is only a `/boot` partition. The `boot` under the Flags column indicates that the partition is bootable. The rest of the disk is consumed by the LVM partition, which is ultimately used to create logical volumes.

For the moment, I recommend that you leave the hard disk alone and find a USB flash drive that you do not mind erasing. You can try the commands I demonstrate on that drive.

Creating a single-partition disk

To add a new storage medium (hard disk, USB flash drive, or similar device) to your computer so that it can be used by Linux, you first need to connect the disk device to your computer and then partition the disk. Here's the general procedure:

1. Install the new hard drive or insert the new USB flash drive.
2. Partition the new disk.
3. Create the filesystems on the new disk.
4. Mount the filesystems.

The easiest way to add a disk or flash drive to Linux is to have the entire disk devoted to a single Linux partition. You can have multiple partitions, however, and assign them each to different types of filesystems and different mount points if you like.

The following process takes you through partitioning a USB flash drive to be used for Linux that has only one partition. If you have a USB flash drive (any size) that you don't mind erasing, you can work through this procedure as you read. The section following this describes how to partition a disk with multiple partitions.

Warning

If you make a mistake partitioning your disk with `parted`, make sure that you correct that change. Unlike `fdisk`, where you could just type **q** to exit without saving your changes, `parted` makes your changes immediately, so you are not able just to quit to abandon changes.

1. For a USB flash drive, just plug it into an available USB port. Going forward, I use a 128GB USB flash drive, but you can get a USB flash drive of any size.
2. Determine the device name for the USB drive. As root user from a shell, type the following `journalctl` command, and then insert the USB flash drive. Messages appear, indicating the device name of the drive you just plugged in (press Ctrl+C to exit the `tail` command when you are finished):

```
# journalctl -f
kernel: usb 4-1: new SuperSpeed Gen 1 USB device number 3 using
                 xhci_hcd
kernel: usb 4-1: New USB device found, idVendor=0781,
                 idProduct=5581, bcdDevice= 1.00
kernel: usb 4-1: New USB device strings: Mfr=1, Product=2,
                 SerialNumber=3
kernel: usb 4-1: Product: Ultra
kernel: usb 4-1: Manufacturer: SanDisk
...
kernel: sd 6:0:0:0: Attached scsi generic sg2 type 0
kernel:  sdb: sdb1
kernel: sd 6:0:0:0: [sdb] Attached SCSI removable disk
udisksd[809]: Mounted /dev/sdb1 at /run/media/chris/7DEB-B010
              on behalf of uid 1000
```

3. From the output, you can see that the USB flash drive was found and assigned to `/dev/sdb`. (Your device name may be different.) It also contains a single formatted partition: `sdb1`. Be sure you identify the correct disk or you could lose all data from disks you may want to keep!
4. If the USB flash drive mounts automatically, unmount it. Here is how to find the USB partitions in this example and unmount them:

```
# mount | grep sdb
/dev/sdb1 on /run/media...
# umount /dev/sdb1
```

5. Use the `parted` command to create partitions on the USB drive. For example, if you are formatting the second USB, SATA, or SCSI disk (`sdb`), you can type the following:

```
# parted /dev/sdb
GNU Parted 3.2
Using /dev/sdb
Welcome to GNU Parted! Type 'help' to view a list of commands.
(parted)
```

   Now you are in `parted` command mode, where you can use the `parted` single-letter command set to work with your partitions.

6. If you start with a new USB flash drive, it may have one partition that is entirely devoted to a Windows-compatible filesystem (such as VFAT or fat32). Use `p` to view all partitions and `rm` to delete the partition. Here's what it looked like when I did that:

```
(parted) p
Model: SanDisk Ultra (scsi)
Disk /dev/sdb: 123GB
Sector size (logical/physical): 512B/512B
Partition Table: msdos
Disk Flags:

Number  Start   End    Size   Type     File system  Flags
1      16.4kB  123GB  123GB  primary  fat32        lba
(parted) rm
Partition number? 1
```

7. Relabel the disk as having a gpt partition table.

```
(parted) mklabel gpt
Warning: The existing disk label on /dev/sdb will be destroyed and all data
on this disk will be lost. Do you want to continue?
Yes/No? Yes
(parted)
```

8. To create a new partition, type **`mkpart`**. You are prompted for the file system type, then the start and end of the partition. This example names the partition `alldisk`, uses `xfs` as the file system type, starts the partition at 1M and ends at 123GB:

```
(parted) mkpart
Partition name?  []? alldisk
File system type?  [ext2]? xfs
Start? 1
End? 123GB
```

9. Double-check that the drive is partitioned the way you want by pressing **`p`**. (Your output will differ, depending on the size of your drive.)

```
(parted) p
Model: SanDisk Ultra (scsi)
Disk /dev/sdb: 123GB
Sector size (logical/physical): 512B/512B
Partition Table: gpt
Disk Flags:

Number  Start   End    Size   File system  Name     Flags
 1      1049kB  123GB  123GB  xfs          alldisk
```

10. Although the partitioning is done, the new partition is not yet ready to use. For that, you have to create a filesystem on the new partition. To create a filesystem on the new disk partition, use the `mkfs` command. By default, this command creates an `ext2` filesystem, which is usable by Linux. However, in most cases you want to use a journaling filesystem (such as ext3, ext4, or xfs). To create an xfs filesystem on the first partition of the second hard disk, type the following:

```
# mkfs -t xfs /dev/sdb1
```

TIP

You can use different commands, or options to this command, to create other filesystem types. For example, use `mkfs.exfat` to create a VFAT filesystem, `mkfs.msdos` for DOS, or `mkfs.ext4` for the ext4 filesystem type. You may want a VFAT or exFAT (available with Ubuntu) filesystem if you want to share files among Linux, Windows, and Mac systems.

11. To be able to use the new filesystem, you need to create a mount point and mount it to the partition. Here is an example of how to do that. You then check to make sure that the mount succeeded.

```
# mkdir /mnt/test
# mount /dev/sdb1 /mnt/test
# df -h /mnt/sdb1
Filesystem         Size  Used Avail Use% Mounted on
/dev/sdb1          115G   13M  115G   1% /mnt/test
```

The `df` command shows that `/dev/sdb1` is mounted on `/mnt/test` and that it offers about 115GB of disk space. The `mount` command shows all mounted filesystems, but here I just list `sdb1` to show that it is mounted.

Any files or directories that you create later in the `/mnt/test` directory, and any of its subdirectories, are stored on the `/dev/sdb1` device.

12. When you are finished using the drive, you can unmount it with the `umount` command, after which you can safely remove the drive (see the description of the `umount` command later if this command fails):

```
# umount /dev/sdb1
```

13. You don't usually set up a USB flash drive to mount automatically every time the system boots because it mounts automatically when you plug it in. But if you decide that you want to do that, edit `/etc/fstab` and add a line describing what and where to mount. Here is an example of a line you might add:

```
/dev/sdb1     /mnt/test       xfs     defaults     0 1
```

In this example, the partition (`/dev/sdb1`) is mounted on the `/mnt/test` directory as an `xfs` filesystem. The `defaults` keyword causes the partition to be

mounted at boot time. The number 0 tells the system not to back up files automatically from this filesystem with the `dump` command (`dump` is rarely used anymore, but the field is here). The `1` in the last column tells the system to check the partition for errors after a certain number of mounts.

At this point, you have a working, permanently mounted disk partition. The next section describes how to partition a disk that has multiple partitions.

Creating a multiple-partition disk

Now that you understand the basic process of partitioning a disk, adding a filesystem, and making that filesystem available (temporarily and permanently), it is time to try a more complex example. Taking that same 128GB USB flash drive, I ran the procedure described later in this section to create multiple partitions on one disk.

In this procedure, I configure a Master Boot Record (MBR) partition to illustrate how extended partitions work and to use the older `fdisk` command. I create two partitions of 5GB (`sdb1` and `sdb2`), two 3GB (`sdb3` and `sdb5`), and 4GB (`sdb6`). The `sdb4` device is an extended partition, which consumes all remaining disk space. Space from the `sdb5` and `sdb6` partitions is taken from the extended partition. This leaves plenty of space to create new partitions.

As before, insert the USB flash drive and determine the device name (in my case, `/dev/sdb`). Also, be sure to unmount any partitions that mount automatically when you insert the USB flash drive.

Tip

When you indicate the size of each partition, type the plus sign and the number of megabytes or gigabytes you want to assign to the partition. For example, `+1024M` to create a 1024-megabyte partition or `+10G` for a 10-gigabyte partition. Be sure to remember the plus sign (`+`) and the `M` or `G`! If you forget the `M` or `G`, `fdisk` thinks you mean sectors and you get unexpected results.

1. I started this procedure by overwriting the USB drive with the `dd` command (`dd if=/dev/zero of=/dev/sd<number> bs=1M count=100`). This allowed me to start with a fresh master boot record. Please be careful to use the right drive number, or you could erase your operating system!
2. Create six new partitions as follows.

```
# fdisk /dev/sdb
Welcome to fdisk (util-linux 2.33.2).
Changes will remain in memory only, until you decide to write them.
Be careful before using the write command.

Device does not contain a recognized partition table.
Created a new DOS disklabel with disk identifier 0x8933f665.
```

```
        Command (m for help): n
        Partition type
           p   primary (0 primary, 0 extended, 4 free)
           e   extended (container for logical partitions)
        Select (default p): p
        Partition number (1-4, default 1): 1
        First sector (2048-240254975, default 2048):
        Last sector, +/-sectors or +/-size{K,M,G,T,P} (2048-
240254975, default 240254975): +5G

        Created a new partition 1 of type 'Linux' and of size 5 GiB.

        Command (m for help): n
        Partition type
           p   primary (1 primary, 0 extended, 3 free)
           e   extended (container for logical partitions)
        Select (default p): p
        Partition number (2-4, default 2): 2
        First sector (10487808-240254975, default 10487808):
        Last sector, +/-sectors or +/-size{K,M,G,T,P} (10487808-
240254975, default 240254975): +5G

        Created a new partition 2 of type 'Linux' and of size 5 GiB.

        Command (m for help): n
        Partition type
           p   primary (2 primary, 0 extended, 2 free)
           e   extended (container for logical partitions)
        Select (default p): p
        Partition number (3,4, default 3): 3
        First sector (20973568-240254975, default 20973568):
        Last sector, +/-sectors or +/-size{K,M,G,T,P} (20973568-
240254975, default 240254975): +3G

        Created a new partition 3 of type 'Linux' and of size 3 GiB.

        Command (m for help): n
        Partition type
           p   primary (3 primary, 0 extended, 1 free)
           e   extended (container for logical partitions)
        Select (default e): e
        Selected partition 4
```

```
        First sector (27265024-240254975, default 27265024):
        Last sector, +/-sectors or +/-size{K,M,G,T,P} (27265024-
240254975, default 240254975): <ENTER>

        Created a new partition 4 of type 'Extended' and of size 101.6 GiB.

        Command (m for help): n
        All primary partitions are in use.
        Adding logical partition 5
        First sector (27267072-240254975, default 27267072):
        Last sector, +/-sectors or +/-size{K,M,G,T,P} (27267072-
240254975, default 240254975): +3G

        Created a new partition 5 of type 'Linux' and of size 3 GiB.

        Command (m for help): n
        All primary partitions are in use.
        Adding logical partition 6
        First sector (33560576-240254975, default 33560576):
        Last sector, +/-sectors or +/-size{K,M,G,T,P} (33560576-
240254975, default 240254975): +4G

        Created a new partition 6 of type 'Linux' and of size 4 GiB.
```

3. Check the partitioning before saving by typing **p**. Notice that there are five usable partitions (`sdc1`, `sdc2`, `sdc3`, `sdc5`, and `sdc6`) and that the sectors between the Start and End for `sdc4` are being consumed by `sdc5` and `sdc6`.

```
        Command (m for help): p
        ...
       Device     Boot    Start       End   Sectors   Size Id Type
       /dev/sdb1           2048  10487807  10485760     5G 83 Linux
       /dev/sdb2       10487808  20973567  10485760     5G 82 Linux
       /dev/sdb3       20973568  27265023   6291456     3G 83 Linux
       /dev/sdb4       27265024 240254975 212989952 101.6G  5 Extended
       /dev/sdb5       27267072  33558527   6291456     3G 83 Linux
       /dev/sdb6       33560576  41949183   8388608     4G 83 Linux
```

4. The default partition type is Linux. I decided that I wanted to use some of the partitions for swap space (`type 82`), FAT32 (`type x`), and Linux LVM (`type 8e`). To do that, I type **t** and indicate which partition type to use. Type **L** to see a list of partition types.

```
        Command (m for help): t
        Partition number (1-6): 2
        Hex code (type L to list codes): 82
```

```
Changed type of partition 'Linux' to 'Linux swap / Solaris'.

Command (m for help): t

Partition number (1-6): 5
Hex code (type L to list codes): c
Changed type of partition 'Linux' to 'W95 FAT32 (LBA)'.

Command (m for help): t

Partition number (1-6): 6
Hex code (type L to list codes): 8e
Changed type of partition 'Linux' to 'Linux LVM'.
```

5. I check that the partition table is the way I want it and then write the changes:

```
Command (m for help): p
...
Device     Boot    Start       End   Sectors   Size Id Type
/dev/sdb1           2048  10487807  10485760     5G 83 Linux
/dev/sdb2       10487808  20973567  10485760     5G 82 Linux swap / Solaris
/dev/sdb3       20973568  27265023   6291456     3G 83 Linux
/dev/sdb4      27265024  240254975 212989952 101.6G  5 Extended
/dev/sdb5      27267072  33558527    6291456     3G  c W95 FAT32 (LBA)
/dev/sdb6    33560576  41949183 8388608    4G 8e Linux LVM

Command (m for help): w

The partition table has been altered!
The kernel still uses the old partitions. The new table
will be used at the next reboot.
Syncing disks
```

6. After the write is completed, check that the kernel knows about the changes to the partition table. To do that, search the `/proc/partitions` for `sdb`. If the new devices are not there, run the `partprobe /dev/sdb` command on the drive or reboot your computer.

```
# grep sdb /proc/partitions
   8       16  120127488 sdb
   8       17  120125440 sdb1
# partprobe /dev/sdb
# grep sdb /proc/partitions
   8       16  120127488 sdb
   8       17    5242880 sdb1
   8       18    5242880 sdb2
   8       19    3145728 sdb3
   8       20          1 sdb4
   8       21    3145728 sdb5
   8       22    4194304 sdb6
```

7. While the partitions are now set for different types of content, other commands are needed to structure the partitions into filesystems or swap areas. Here's how to do that for the partitions just created:

   sdb1: To make this into a regular Linux `ext4` filesystem, type the following:

   ```
   # mkfs -t ext4 /dev/sdb1
   ```

   sdb2: To format this as a swap area, type the following:

   ```
   # mkswap /dev/sdb2
   ```

   sdb3: To make this into an `ext2` filesystem (the default), type the following:

   ```
   # mkfs /dev/sdb3
   ```

   sdb5: To make this into a VFAT filesystem (the default), type the following:

   ```
   # mkfs -t vfat /dev/sdb5
   ```

   sdb6: To make this into a LVM physical volume, type the following:

   ```
   # pvcreate /dev/sdb6
   ```


These partitions are now ready to be mounted, used as swap areas, or added to an LVM volume group. See the next section, "Using Logical Volume Manager Partitions," to see how LVM physical volumes are used to ultimately create LVM logical volumes from volume groups. See the section "Mounting Filesystems" for descriptions of how to mount filesystems and enable swap areas.

Using Logical Volume Manager Partitions

Basic disk partitioning in Linux has its shortcomings. What happens if you run out of disk space? In the old days, a common solution was to copy data to a bigger disk, restart the system with the new disk, and hope that you didn't run out of space again anytime soon. This process meant downtime and inefficiency.

Logical Volume Manager (LVM) offers lots of flexibility and efficiency in dealing with constantly changing storage needs. With LVM, physical disk partitions are added to pools of space called volume groups. Logical volumes are assigned space from volume groups as needed. This gives you these abilities:

- Add more space to a logical volume from the volume group while the volume is still in use.
- Add more physical volumes to a volume group if the volume group begins to run out of space. The physical volumes can be from disks.
- Move data from one physical volume to another so you can remove smaller disks and replace them with larger ones while the filesystems are still in use—again, without downtime.

With LVM, it is also easier to shrink filesystems to reclaim disk space, although shrinking does require that you unmount the logical volume (but no reboot is needed). LVM also supports advanced features, such as mirroring and working in clusters.

Checking an existing LVM

Let's start by looking at an existing LVM example on a Red Hat Enterprise Linux system. The following command displays the partitions on my first hard disk:

```
# fdisk -l /dev/sda | grep /dev/sda
Disk /dev/sda: 160.0 GB, 160000000000 bytes
/dev/sda1   *        2048     1026047      512000    83   Linux
/dev/sda2   *     1026048   312498175   155736064    8e   Linux LVM
```

On this RHEL system, the 160GB hard drive is divided into one 500MB Linux partition (`sda1`) and a second (Linux LVM) partition that consumes the rest of the disk (`sda2`). Next, I use the `pvdisplay` command to see if that partition is being used in an LVM group:

```
# pvdisplay /dev/sda2
  --- Physical volume ---
  PV Name               /dev/sda2
  VG Name               vg_abc
  PV Size               148.52 GiB / not usable 2.00 MiB
  Allocatable           yes (but full)
  PE Size               4.00 MiB
  Total PE              38021
  Free PE               0
  Allocated PE          38021
  PV UUID               wlvuIv-UiI2-pNND-f39j-oH0X-9too-AOII7R
```

You can see that the LVM physical volume represented by `/dev/sda2` has 148.52GiB of space, all of which has been totally allocated to a volume group named `vg_abc`. The smallest unit of storage that can be used from this physical volume is 4.0MiB, which is referred to as a *Physical Extent (PE)*.

Note

Notice that LVM tools show disk space in MiB and GiB. One MB is 1,000,000 bytes (10∧6), while a MiB is 1,048,576 bytes (2∧20). A MiB is a more accurate way to reflect how data are stored on a computer. But marketing people tend to use MB because it makes the hard disks, CDs, and DVDs they sell look like they have more capacity than they do. Keep in mind that most tools in Linux display storage data in MiB and GiB, although some can display MB and GB as well.

Next, you want to see information about the volume group:

```
# vgdisplay vg_abc
  --- Volume group ---
  VG Name               vg_abc
```

```
  System ID
  Format                lvm2
  Metadata Areas        1
  Metadata Sequence No  4
  VG Access             read/write
  VG Status             resizable
  MAX LV                0
  Cur LV                3
  Open LV               3
  Max PV                0
  Cur PV                1
  Act PV                1
  VG Size               148.52 GiB
  PE Size               4.00 MiB
  Total PE              38021
  Alloc PE / Size       38021 / 148.52 GiB
  Free  PE / Size       0 / 0
  VG UUID               c2SGHM-KU9H-wbXM-sgca-EtBr-UXAq-UnnSTh
```


You can see that all of the 38,021 PEs have been allocated. Using `lvdisplay` as follows, you can see where they have been allocated (I have snipped some of the output):

```
# lvdisplay vg_abc
  --- Logical volume ---
  LV Name                /dev/vg_abc/lv_root
  VG Name                vg_abc
  LV UUID                33VeDc-jd0l-hlCc-RMuB-tkcw-QvFi-cKCZqa
  LV Write Access        read/write
  LV Status              available
  # open                 1
  LV Size                50.00 GiB
  Current LE             12800
  Segments               1
  Allocation             inherit
  Read ahead sectors     auto
  - currently set to     256
  Block device           253:0
  --- Logical volume ---
  LV Name                /dev/vg_abc/lv_home
  VG Name                vg_abc
  ...
  LV Size                92.64 GiB
  --- Logical volume ---
  LV Name                /dev/vg_abc/lv_swap
  VG Name                vg_abc
  ...
  LV Size                5.88 GiB
```

There are three logical volumes drawing space from `vg_abc`. Each logical volume is associated with a device name that includes the volume group name and the logical volume

name: /dev/vg_abc/lv_root (50GB), /dev/vg_abc/lv_home (92.64GB), and /dev/vg_abc/lv_swap (5.88GB). Other devices linked to these names are located in the /dev/mapper directory: vg_abc-lv_home, vg_abc-lv_root, and vg_abc-lv_swap. Either set of names can be used to refer to these logical volumes.

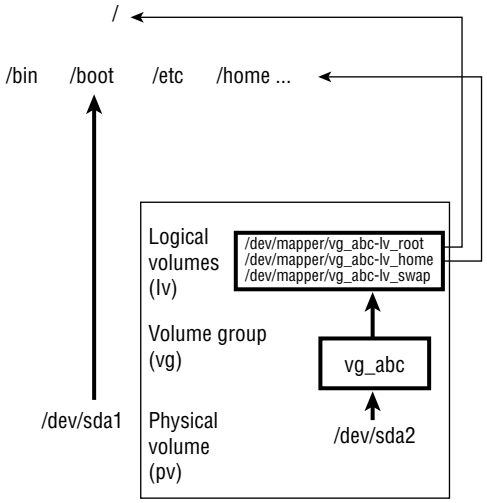
The root and home logical volumes are formatted as ext4 filesystems, whereas the swap logical volume is formatted as swap space. Let's look in the /etc/fstab file to see how these logical volumes are used:

```
# grep vg_ /etc/fstab
/dev/mapper/vg_abc-lv_root /      ext4  defaults    1 1
/dev/mapper/vg_abc-lv_home /home  ext4  defaults    1 2
/dev/mapper/vg_abc-lv_swap swap   swap  defaults    0 0
```

Figure 12.1 illustrates how the different partitions, volume groups, and logical volumes relate to the complete Linux filesystem. The sda1 device is formatted as a filesystem and mounted on the /boot directory. The sda2 device provides space for the vg_abc volume group. Then logical volumes lv_home and lv_root are mounted on the /home and / directories, respectively.

FIGURE 12.1

LVM logical volumes can be mounted like regular partitions on a Linux filesystem.



If you run out of space on any of the logical volumes, you can assign more space from the volume group. If the volume group is out of space, you can add another hard drive or network storage drive and add space from that drive to the volume group so more is available.

Now that you know how LVM works, the next section shows you how to create LVM logical volumes from scratch.

Creating LVM logical volumes

LVM logical volumes are used from the top down, but they are created from the bottom up. As illustrated in Figure 12.1, first you create one or more physical volumes (pv), use the physical volumes to create volume groups (vg), and then create logical volumes from the volume groups (lv).

Commands for working with each LVM component begin with the letters `pv`, `vg`, and `lv`. For example, `pvdisplay` shows physical volumes, `vgdisplay` shows volume groups, and `lvdisplay` shows logical volumes.

The following procedure takes you through the steps of creating LVM volumes from scratch. To do this procedure, you could use the USB flash drive and partitions that I described earlier in this chapter.

1. Obtain a disk with some spare space on it and create a disk partition on it of the LVM type (`8e`). Then use the `pvcreate` command to identify this partition as an LVM physical volume. The process of doing this is described in the section "Creating a multiple-partition disk" using the `/dev/sdb6` device in that example.
2. To add that physical volume to a new volume group, use the `vgcreate` command. The following command shows you how to create a volume group called `myvg0` using the `/dev/sdb6` device:

```
# vgcreate myvg0 /dev/sdc6
    Volume group "myvg0" successfully created
```

3. To see the new volume group, type the following:

```
# vgdisplay myvg0
  --- Volume group ---
  VG Name               myvg0
  ...
  VG Size               <4.00 GiB
  PE Size               4.00 MiB
  Total PE              1023
  Alloc PE / Size       0 / 0
  Free  PE / Size       1023 / <4.00 MiB
```

4. All of the 1023 physical extents (PEs, 4.00 MiB each) are available. Here's how to create a logical volume from some of the space in that volume group and then check that the device for that logical volume exists:

```
# lvcreate -n music -L 1G myvg0
  Logical volume "music" created
# ls /dev/mapper/myvg0*
/dev/mapper/myvg0-music
```

5. As you can see, the procedure created a device named `/dev/mapper/myvg0-music`. That device can now be used to put a filesystem on and mount, just as you did with regular partitions in the first part of this chapter. For example:

```
# mkfs -t ext4 /dev/mapper/myvg0-music
# mkdir /mnt/mymusic
# mount /dev/mapper/myvg0-music /mnt/mymusic
# df -h /mnt/mymusic
Filesystem               Size  Used  Avail  Use%  Mounted on
/dev/mapper/myvg0-music  976M  2.6M   987M    1%  /mnt/mymusic
```

6. As with regular partitions, logical volumes can be mounted permanently by adding an entry to the `/etc/fstab` file, such as

```
/dev/mapper/myvg0-music /mnt/mymusic  ext4 defaults 1 2
```

The next time you reboot, the logical volume is automatically mounted on `/mnt/mymusic`. (Be sure to unmount the logical volume and remove this line if you want to remove the USB flash drive from your computer.)

Growing LVM logical volumes

If you run out of space on a logical volume, you can add space to it without even unmounting it. To do that, you must have space available in the volume group, grow the logical volume, and grow the filesystem to fill it. Building on the procedure in the previous section, here's how to grow a logical volume:

1. Note how much space is currently on the logical volume, and then check that space is available in the logical volume's volume group:

```
# vgdisplay myvg0
...
  VG Size               <4.00 MiB
  PE Size               4.00 MiB
  Total PE              1023
  Alloc PE / Size       256 / 1.00 GiB
  Free  PE / Size       767 / <3.00 GiB
# df -h /mnt/mymusic/
Filesystem                 Size  Used Avail Use% Mounted on
/dev/mapper/myvg0-music    976M  2.6M  987M   1% /mnt/mymusic
```

2. Expand the logical volume using the `lvextend` command:

```
# lvextend -L +1G /dev/mapper/myvg0-music
  Size of logical volume myvg0/music changed
         from 1.00GiB to 2.00 GiB (512 extents).
  Logical volume myvg0/music successfully resized
```

3. Resize the filesystem to fit the new logical volume size:

```
# resize2fs -p /dev/mapper/myvg0-music
```

4. Check to see that the filesystem is now resized to include the additional disk space:

```
# df -h /mnt/mymusic/
Filesystem               Size  Used Avail Use% Mounted on
/dev/mapper/myvg0-music  2.0G  3.0M  1.9G   1% /mnt/mymusic
```

You can see that the filesystem is now about 1G larger.

Mounting Filesystems

Now that you have had a chance to play with disk partitioning and filesystems, I'm going to step back and talk about how filesystems are set up to connect permanently to your Linux system.

Most of the hard disk partitions created when you install Linux are mounted automatically for you when the system boots. When you install Fedora, Ubuntu, Red Hat Enterprise Linux, and other Linux systems, you have the option to let the installer automatically configure your hard disk or create partitions yourself and indicate the mount points for those partitions.

When you boot Linux, usually all of the Linux partitions on your hard disk are listed in your `/etc/fstab` file and are mounted. For that reason, the following sections describe what you might expect to find in that file. It also describes how you can mount other partitions so that they become part of your Linux filesystem.

The `mount` command is used not only to mount local storage devices but also to mount other kinds of filesystems on your Linux system. For example, `mount` can be used to mount directories (folders) over the network from NFS or Samba servers. It can be used to mount filesystems from a new hard drive or USB flash drive that is not configured to automount. It can also mount filesystem image files using loop devices.

Note

With the addition of automatic mounting features and changes in how removable media are identified with the Linux 2.6 kernel (using features such as Udev and Hardware Abstraction Layer), you no longer need to mount removable media manually for many Linux desktop systems. Understanding how to mount and unmount filesystems manually on a Linux server, however, can be a very useful skill if you want to mount remote filesystems or temporarily mount partitions in particular locations.

Supported filesystems

To see filesystem types that are currently loaded in your kernel, type **`cat /proc/filesystems`**. The list that follows shows a sample of filesystem types that are currently supported in Linux, although they may not be in use at the moment or even available on the Linux distribution you are using.

befs: Filesystem used by the BeOS operating system.

btrfs: A copy-on-write filesystem that implements advanced filesystem features. It offers fault tolerance and easy administration. The btrfs file system has recently grown in popularity for enterprise applications.

cifs: Common Internet Filesystem (CIFS), the virtual filesystem used to access servers that comply with the SNIA CIFS specification. CIFS is an attempt to refine and standardize the SMB protocol used by Samba and Windows file sharing.

ext4: Successor to the popular ext3 filesystem. It includes many improvements over ext3, such as support for volumes up to 1 exbibyte and file sizes up to 16 tebibytes. (This replaced ext3 as the default filesystem used in Fedora and RHEL. It has since been supplanted by xfs as the default for RHEL.)

ext3: Ext filesystems are the most common in most Linux systems. Compared ext2, the ext3 filesystem, also called the third extended filesystem, includes journaling features that, compared to ext2, improve a filesystem's capability to recover from crashes.

ext2: The default filesystem type for earlier Linux systems. Features are the same as ext3, except that ext2 doesn't include journaling features.

ext: This is the first version of ext3. It is not used very often anymore.

iso9660: Evolved from the High Sierra filesystem (the original standard for CD-ROMs). Extensions to the High Sierra standard (called Rock Ridge extensions) allow iso9660 filesystems to support long filenames and UNIX-style information (such as file permissions, ownership, and links). Data CD-ROMs typically use this filesystem type.

kafs: AFS client filesystem. Used in distributed computing environments to share files with Linux, Windows, and Macintosh clients.

minix: Minix filesystem type, used originally with the Minix version of UNIX. It supports filenames of up to only 30 characters.

msdos: An MS-DOS filesystem. You can use this type to mount media that comes from old Microsoft operating systems.

vfat: Microsoft extended FAT (VFAT) filesystem.

exfat: Extended FAT (exFAT) file system that has been optimized for SD cards, USB drives, and other flash memory.

umsdos: An MS-DOS filesystem with extensions to allow features that are similar to UNIX (including long filenames).

proc: Not a real filesystem, but rather a filesystem interface to the Linux kernel. You probably won't do anything special to set up a proc filesystem. However, the `/proc` mount point should be a proc filesystem. Many utilities rely on `/proc` to gain access to Linux kernel information.

reiserfs: ReiserFS journaled filesystem. ReiserFS was once a common default filesystem type for several Linux distributions. However, ext and xfs filesystems are by far more common filesystem types used with Linux today.

swap: Used for swap partitions. Swap areas are used to hold data temporarily when RAM is used up. Data is swapped to the swap area and then returned to RAM when it is needed again.

squashfs: Compressed, read-only filesystem type. Squashfs is popular on live CDs, where there is limited space and a read-only medium (such as a CD or DVD).

nfs: Network Filesystem (NFS) type of filesystem. NFS is used to mount filesystems on other Linux or UNIX computers.

hpfs: Filesystem is used to do read-only mounts of an OS/2 HPFS filesystem.

ncpfs: A filesystem used with Novell NetWare. NetWare filesystems can be mounted over a network.

ntfs: Windows NT filesystem. Depending upon the distribution you have, it may be supported as a read-only filesystem (so that you can mount and copy files from it).

ufs: Filesystem popular on Sun Microsystems's operating systems (that is, Solaris and SunOS).

jfs: A 64-bit journaling filesystem by IBM that is relatively lightweight for the many features it has.

xfs: A high-performance filesystem originally developed by Silicon Graphics that works extremely well with large files. This filesystem is the default type for RHEL 7.

gfs2: A shared disk filesystem that allows multiple machines to all use the same shared disk without going through a network filesystem layer such as CIFS, NFS, and so on.

To see the list of filesystems that come with the kernel you are using, type **`ls /lib/modules/kernelversion/kernel/fs/`**. The actual modules are stored in subdirectories of that directory. Mounting a filesystem of a supported type causes the filesystem module to be loaded, if it is not already loaded.

Type **`man fs`** to see descriptions of Linux filesystems.

Enabling swap areas

A *swap area* is an area of the disk that is made available to Linux if the system runs out of memory (RAM). If your RAM is full and you try to start another application without a swap area, that application will fail.

With a swap area, Linux can temporarily swap out data from RAM to the swap area and then get it back when needed. You take a performance hit, but it is better than having processes fail.

To create a swap area from a partition or a file, use the `mkswap` command. To enable that swap area temporarily, you can use the `swapon` command. For example, here's how to check your available swap space, create a swap file, enable the swap file, and then check that the space is available on your system:

```
# free -m
         total    used     free    shared  buffers  cached
Mem:      1955     663     1291         0       42     283
-/+ buffers/cache:          337      1617
Swap:      819       0      819
# dd if=/dev/zero of=/var/tmp/myswap bs=1M count=1024
# mkswap /var/opt/myswap
# swapon /var/opt/myswap
# free -m
         total    used     free  shared  buffers     cached
Mem:      1955    1720      235       0       42       1310
-/+ buffers/cache:          367    1588
Swap:     1843       0     1843
```

The `free` command shows the amount of swap before and after creating, making, and enabling the swap area with the `swapon` command. That amount of swap is available immediately and temporarily to your system. To make that swap area permanent, you need to add it to your `/etc/fstab` file. Here is an example:

```
/var/opt/myswap  swap   swap    defaults   0 0
```

This entry indicates that the swap file named `/var/opt/myswap` should be enabled at boot time. Because there is no mount point for swap area, the second field is just set to swap, as is the partition type. To test that the swap file works before rebooting, you can enable it immediately (`swapon -a`) and check that the additional swap area appears:

```
# swapon -a
```

Disabling swap area

If at any point you want to disable a swap area, you can do so using the `swapoff` command. You might do this, in particular, if the swap area is no longer needed and you want to reclaim the space being consumed by a swap file or remove a USB drive that is providing a swap partition.

First, make sure that no space is being used on the swap device (using the `free` command), and then use `swapoff` to turn off the swap area so that you can reuse the space. Here is an example:

```
# free -m
         total    used     free     shared    buffers     cached
Mem:      1955    1720      235          0         42       1310
-/+ buffers/cache: 367     1588
```

```
Swap:     1843       0     1843
# swapoff /var/opt/myswap
# free -m
Mem:      1955    1720      235          0         42       1310
-/+ buffers/cache: 367     1588
Swap:      819       0      819
```

Notice that the amount of available swap was reduced after running the `swapoff` **command.**

Using the fstab file to define mountable file systems

The hard disk partitions on your local computer and the remote filesystems that you use every day are probably set up to mount automatically when you boot Linux. The `/etc/fstab` file contains definitions for each partition, along with options describing how the partition is mounted. Here's an example of an `/etc/fstab` file:

```
# /etc/fstab
/dev/mapper/vg_abc-lv_root     /        ext4   defaults       1 1
UUID=78bdae46-9389-438d-bfee-06dd934fae28 /boot ext4  defaults  1 2
/dev/mapper/vg_abc-lv_home     /home    ext4   defaults       1 2
/dev/mapper/vg_abc-lv_swap     swap     swap   defaults       0 0
# Mount entries added later
/dev/sdb1                      /win     vfat   ro             1 2
192.168.0.27:/nfsstuff         /remote  nfs    users,_netdev  0 0
//192.168.0.28/myshare         /share   cifs   guest,_netdev  0 0
# special Linux filesystems
tmpfs                          /dev/shm tmpfs   defaults      0 0
devpts                         /dev/pts devpts gid=5,mode=620 0 0
sysfs                          /sys     sysfs  defaults       0 0
proc                           /proc    proc   defaults       0 0
```

The `/etc/fstab` file just shown is from a default Red Hat Enterprise Linux 6 server install, with a few lines added.

For now, you can ignore the `tmpfs`, `devpts`, `sysfs`, and `proc` entries. Those are special devices associated with shared memory, terminal windows, device information, and kernel parameters, respectively.

In general, the first column of `/etc/fstab` shows the device or share (what is mounted), while the second column shows the mount point (where it is mounted). That is followed by the type of filesystem, any mount options (or defaults), and two numbers (used to tell commands such as `dump` and `fsck` what to do with the filesystem).

The first three entries represent the disk partitions assigned to the root of the filesystem (`/`), the `/boot` directory, and the `/home` directory. All three are `ext4` filesystems. The fourth line is a swap device (used to store data when RAM overflows). Notice that the device names for `/`, `/home`, and `swap` all start with `/dev/mapper`. That's because they are LVM logical volumes that are assigned space from a pool of space called an LVM volume group (more on LVM in the section "Using Logical Volume Manager Partitions" earlier in this chapter).

The /boot partition is on its own physical partition, /dev/sda1. Instead of using /dev/sda1, however, a unique identifier (UUID) identifies the device. Why use a UUID instead of /dev/sda1 to identify the device? Suppose you plugged another disk into your computer and booted up. Depending on how your computer iterates through connected devices on boot, it is possible that the new disk might be identified as /dev/sda, causing the system to look for the contents of /boot on the first partition of that disk.

To see all of the UUIDs assigned to storage devices on your system, type the blkid command, as follows:

```
# blkid
/dev/sda1:
  UUID="78bdae46-9389-438d-bfee-06dd934fae28" TYPE="ext4"
/dev/sda2:
  UUID="wlvuIv-UiI2-pNND-f39j-oH0X-9too-AOII7R" TYPE="LVM2_member"
/dev/mapper/vg_abc-lv_root:
  UUID="3e6f49a6-8fec-45e1-90a9-38431284b689" TYPE="ext4"
/dev/mapper/vg_abc-lv_swap:
  UUID="77662950-2cc2-4bd9-a860-34669535619d" TYPE="swap"
/dev/mapper/vg_abc-lv_home:
  UUID="7ffbcff3-36b9-4cbb-871d-091efb179790" TYPE="ext4"
/dev/sdb1:
  SEC_TYPE="msdos" UUID="75E0-96AA" TYPE="vfat"
```

Any of the device names can be replaced by the UUID designation in the left column of an /etc/fstab entry.

I added the next three entries in /etc/fstab to illustrate some different kinds of entries. I connected a hard drive from an old Microsoft Windows system and had it mounted on the /win directory. I added the ro option so it would mount read-only.

The next two entries represent remote filesystems. On the /remote directory, the /nfsstuff directory is mounted read/write (rw) from the host at address 192.168.0.27 as an NFS share. On the /share directory, the Windows share named myshare is mounted from the host at 192.168.0.28. In both cases, I added the _netdev option, which tells Linux to wait for the network to come up before trying to mount the shares. For more information on mounting CIFS and NFS shares, refer to Chapters 19, "Configuring a Windows File Sharing (Samba) Server," and 20, "Configuring an NFS File Server," respectively.

Coming from Windows

The section "Using the fstab file to define mountable file systems" shows mounting a hard disk partition from an old VFAT filesystem being used in Windows. Most Windows systems today use the NTFS filesystem. Support for this system, however, is not delivered with every Linux system. NTFS is available from Fedora in the ntfs-3g package.

To help you understand the contents of the /etc/fstab file, here is what is in each field of that file:

Field 1: Name of the device representing the filesystem. This field can include the LABEL or UUID option with which you can indicate a volume label or universally unique identifier (UUID) instead of a device name. The advantage to this approach is that because the partition is identified by volume name, you can move a volume to a different device name and not have to change the fstab file. (See the description of the mkfs command in the section "Using the mkfs Command to Create a Filesystem" later in this chapter for information on creating and using labels.)

Field 2: Mount point in the filesystem. The filesystem contains all data from the mount point down the directory tree structure unless another filesystem is mounted at some point beneath it.

Field 3: Filesystem type. Valid filesystem types are described in the section "Supported filesystems" earlier in this chapter (although you can only use filesystem types for which drivers are included for your kernel).

Field 4: Use defaults or a comma-separated list of options (no spaces) that you want to use when the entry is mounted. See the mount command manual page (under the -o option) for information on other supported options.

Tip

Typically, only the root user is allowed to mount a filesystem using the mount command. However, to allow any user to mount a filesystem (such as a filesystem on a CD), you could add the user option to Field 4 of /etc/fstab.

Field 5: The number in this field indicates whether the filesystem needs to be dumped (that is, have its data backed up). A 1 means that the filesystem needs to be dumped, and a 0 means that it doesn't. (This field is no longer particularly useful because most Linux administrators use more sophisticated backup options than the dump command. Most often, a 0 is used.)

Field 6: The number in this field indicates whether the indicated filesystem should be checked with fsck when the time comes for it to be checked: 1 means it needs to be checked first, 2 means to check after all those indicated by 1 have already been checked, and 0 means don't check it.

If you want to find out more about mount options as well as other features of the /etc/fstab file, there are several man pages to which you can refer, including man 5 nfs and man 8 mount.

Using the mount command to mount file systems

Linux systems automatically run mount -a (mount all filesystems from the /etc/fstab file) each time you boot. For that reason, you generally use the mount

command only for special situations. In particular, the average user or administrator uses `mount` in two ways:

- To display the disks, partitions, and remote filesystems currently mounted
- To mount a filesystem temporarily

Any user can type `mount` (with no options) to see what filesystems are currently mounted on the local Linux system. The following is an example of the `mount` command. It shows a single hard disk partition (`/dev/sda1`) containing the root (`/`) filesystem and proc and devpts filesystem types mounted on `/proc` and `/dev`, respectively.

```
$ mount
/dev/sda3 on / type ext4 (rw)
/dev/sda2 on /boot type ext4 (rw)
/dev/sda1 on /mnt/win type vfat (rw)
/dev/proc on /proc type proc (rw)
/dev/sys on /sys type sysfs (rw)
/dev/devpts on /dev/pts type devpts (rw,gid=5,mode=620)
/dev/shm on /dev/shm type tmpfs (rw)
none on /proc/sys/fs/binfmt_misc type binfmt_misc (rw)
/dev/cdrom on /media/MyOwnDVD type iso9660 (ro,nosuid,nodev)
```

Traditionally, the most common devices to mount by hand are removable media, such as DVDs or CDs. However, depending on the type of desktop you are using, CDs and DVDs may be mounted for you automatically when you insert them. (In some cases, applications are launched as well when media is inserted. For example, a music player or photo editor may be launched when your inserted USB medium has music or digital images on it.)

Occasionally, however, you may find it useful to mount a filesystem manually. For example, you want to look at the contents of an old hard disk, so you install it as a second disk on your computer. If the partitions on the disk did not automount, you could mount partitions from that disk manually. For example, to mount a read-only disk partition `sdb1` that has an older `ext3` filesystem, you could type this:

```
# mkdir /mnt/temp
# mount -t ext3 -o ro /dev/sdb1 /mnt/tmp
```

Another reason to use the `mount` command is to remount a partition to change its mount options. Suppose that you want to remount `/dev/sdb1` as read/write, but you do not want to unmount it (maybe someone is using it). You could use the remount option as follows:

```
# mount -t ext3 -o remount,rw /dev/sdb1
```

Mounting a disk image in loopback

Another valuable way to use the `mount` command has to do with disk images. If you download an SD card or DVD ISO image file from the Internet and you want to see what it contains, you can do so without burning it to DVD or other medium. With the image on your hard disk, create a mount point and use the `-o loop` option to mount it locally. Here's an example:

```
# mkdir /mnt/mydvdimage
# mount -o loop whatever-i686-disc1.iso /mnt/mydvdimage
```

In this example, the /mnt/mydvdimage directory is created, and then the disk image file (whatever-i686-disc1.iso) residing in the current directory is mounted on it. You can now cd to that directory, view the contents of it, and copy or use any of its contents. This is useful for downloaded DVD images from which you want to install software without having to burn the image to DVD. You could also share that mount point over NFS, so you could install the software from another computer. When you are finished, to unmount the image, type umount /mnt/mydvdimage.

Other options to mount are available only for specific filesystem types. See the mount manual page for those and other useful options.

Using the umount command

When you are finished using a temporary filesystem, or you want to unmount a permanent filesystem temporarily, use the umount command. This command detaches the filesystem from its mount point in your Linux filesystem. To use umount, you can give it either a directory name or a device name, as shown in this example:

```
# umount /mnt/test
```

This unmounts the device from the mount point /mnt/test. You can also unmount using this form:

```
# umount /dev/sdb1
```

In general, it's better to use the directory name (/mnt/test) because the umount command will fail if the device is mounted in more than one location. (Device names all begin with /dev.)

If you get the message device is busy, the umount request has failed because either an application has a file open on the device or you have a shell open with a directory on the device as a current directory. Stop the processes or change to a directory outside the device you are trying to unmount for the umount request to succeed.

An alternative for unmounting a busy device is the -l option. With umount -l (a lazy unmount), the unmount happens as soon as the device is no longer busy. To unmount a remote NFS filesystem that's no longer available (for example, the server went down), you can use the umount -f option to forcibly unmount the NFS filesystem.

Tip

A really useful tool for discovering what's holding open a device you want to unmount is the lsof command. Type lsof with the name of the partition that you want to unmount (such as lsof /mnt/test). The output shows you what commands are holding files open on that partition. The fuser-v /mnt/test command can be used in the same way.

Using the mkfs Command to Create a Filesystem

You can create a filesystem for any supported filesystem type on a disk or partition that you choose. You do so with the `mkfs` command. Although this is most useful for creating filesystems on hard-disk partitions, you can create filesystems on USB flash drives or rewritable DVDs as well.

Before you create a new filesystem, make sure of the following:

- You have partitioned the disk as you want (using the `fdisk` command).
- You get the device name correct, or you may end up overwriting your hard disk by mistake. For example, the first partition on the second SCSI or USB flash drive on your system is `/dev/sdb1` and the third disk is `/dev/sdc1`.
- To unmount the partition if it's mounted before creating the filesystem.

The following are two examples of using `mkfs` to create a filesystem on two partitions on a USB flash drive located as the first and second partitions on the third SCSI disk (`/dev/sdc1` and `/dev/sdc2`). The first creates an xfs partition, while the second creates an ext4 partition.

```
# mkfs -t xfs /dev/sdc1
meta-data=/dev/sda3        isize=256    agcount=4, agsize=256825 blks
         =                 sectsz=512   attr=2, projid32bit=1
         =                 crc=0
data     =                 bsize=4096   blocks=1027300, imaxpct=25
         =                 sunit=0      swidth=0 blks
naming   =version 2        bsize=4096   ascii-ci=0 ftype=0
log      =internal log     bsize=4096   blocks=2560, version=2
         =                 sectsz=512   sunit=0 blks, lazy-count=1
realtime =none             extsz=4096   blocks=0, rtextents=0

# mkfs -t ext4 /dev/sdc2
mke2fs 1.44.6 (5-Mar-2019)
Creating filesystem with 524288 4k blocks and 131072 inodes
Filesystem UUID: 6379d82e-fa25-4160-8ffa-32bc78d410eee
Superblock backups stored on blocks:
        32768, 98304, 163840, 229376, 294912
Allocating group tables: done
Writing inode tables: done
Creating journal (16384 blocks): done
Writing superblocks and filesystem accounting information: done
```

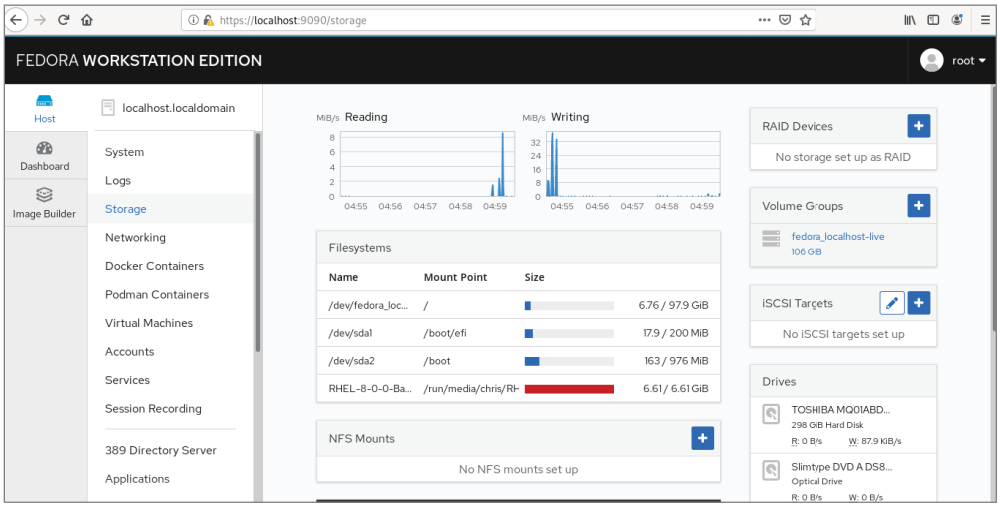
You can now mount either of these filesystems (for example, `mkdir /mnt/myusb ; mount /dev/sdc1 /mnt/myusb`), change to `/mnt/myusb` as your current directory (`cd /mnt/myusb`), and create files on it as you please.

Managing Storage with Cockpit

Most of the features described in this chapter for working with disk partitions and filesystems using command-line tools can be accomplished using the Cockpit web user interface. With Cockpit running on your system, open the web UI (`hostname:9090`) and select the Storage tab. Figure 12.2 shows an example of the Cockpit Storage tab on a Fedora system.

FIGURE 12.2

View storage devices, filesystems, and activities from the Cockpit Storage page.



The Storage tab provides a solid overview of your system's storage. It charts read and write activity of your storage devices every minute. It displays the local filesystems and storage (including RAID devices and LVM volume groups) as well as remotely mounted NFS shares and iSCSI targets. Each hard disk, DVD, and other physical storage device is also displayed on the Storage tab.

Select a mounted filesystem, and you can see and change partitioning for that filesystem. For example, by selecting the entry for a filesystem that was automatically mounted on `/run/media`, you can see all of the partitions for the device it is on (`/dev/sdb1` and `/dev/sdb2`). Figure 12.3 shows that there is an ISO9660 filesystem (typical for bootable media) and a smaller VFAT filesystem on the two partitions.

With the storage device information displayed, you could reformat the entire storage device (Create Partition Table) or, assuming that space is available on the device, add a new partition (Create Partition). Figure 12.4 shows an example of the window that appears when you select Create Partition Table.

FIGURE 12.3

View and change disk partitions for a select storage device.

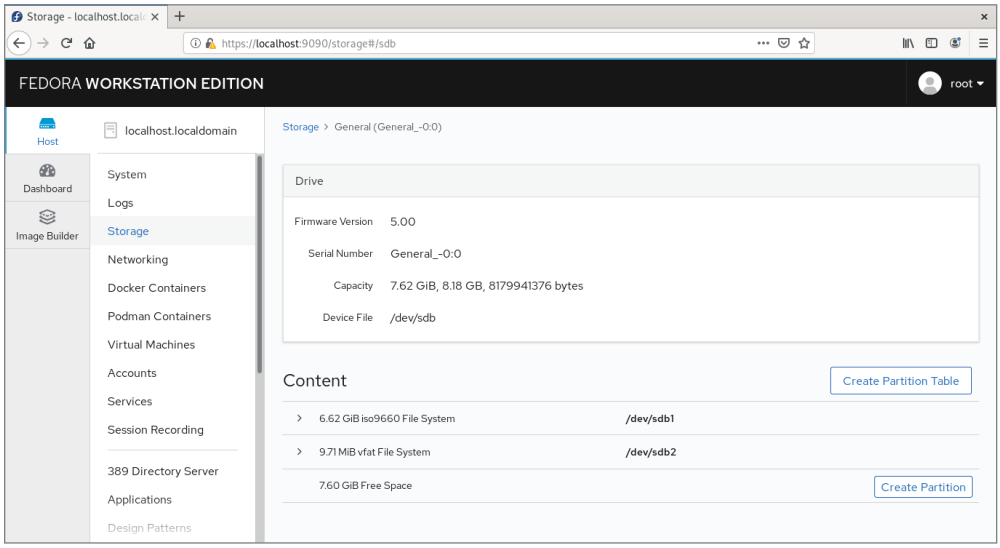
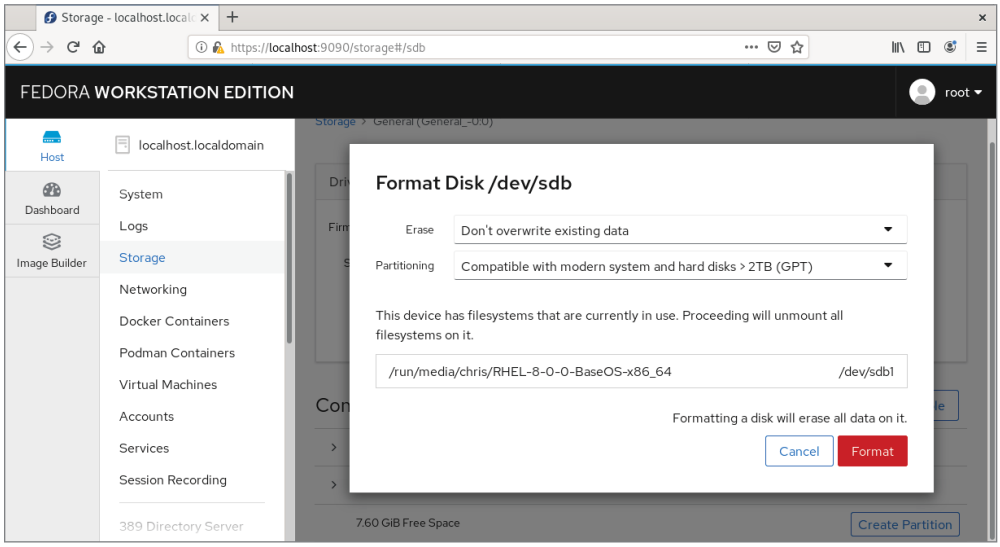


FIGURE 12.4

Creating a new partition table



If you decide that you want to format the disk or USB drive, change the Erase setting to allow all of the data on the drive to be overwritten and then choose the type of partitioning. Select Format to unmount any mounted partitions from the drive and create a new partition table. After that, you can add partitions to the storage device, choosing the size, filesystem type, and whether or not to encrypt data. You can even choose where in the operating system's filesystem to mount the new partition. With just a few selections, you can quickly create the disk layouts that you want in ways that are more intuitive than methods for doing comparable steps from the command line.

Summary

Managing filesystems is a critical part of administering a Linux system. Using commands such as `fdisk`, you can view and change disk partitions. Filesystems can be added to partitions using the `mkfs` command. Once created, filesystems can be mounted and unmounted using the `mount` and `umount` commands, respectively.

Logical Volume Manager (LVM) offers a more powerful and flexible way of managing disk partitions. With LVM, you create pools of storage, called volume groups, which can allow you to grow and shrink logical volumes as well as extend the size of your volume groups by adding more physical volumes.

For a more intuitive way of working with storage devices, Cockpit offers an intuitive, Web-based interface for viewing and configuring storage on your Linux system. Using the Web UI, you can see both local and networked storage as well as reformat disks and modify disk partitions.

With most of the basics needed to become a system administrator covered at this point in the book, Chapter 13, "Understanding Server Administration," introduces concepts for extending those skills to manage network servers. Topics in that chapter include information on how to install, manage, and secure servers.

Exercises

Use these exercises to test your knowledge of creating disk partitions, Logical Volume Manager, and working with filesystems. You need a USB flash drive that is at least 1GB, which you can erase for these exercises.

These tasks assume that you are running a Fedora or Red Hat Enterprise Linux system (although some tasks work on other Linux systems as well). If you are stuck, solutions to the tasks are shown in Appendix B (although in Linux, there are often multiple ways to complete a task).

1. Run a command as root to watch the system journal in a Terminal as fresh data comes in and insert your USB flash drive. Determine the device name of the USB flash drive.

2. Run a command to list the partition table for the USB flash drive.
3. Delete all the partitions on your USB flash drive, save the changes, and make sure the changes were made both on the disk's partition table and in the Linux kernel.
4. Add three partitions to the USB flash drive: 100MB Linux partition, 200MB swap partition, and 500MB LVM partition. Save the changes.
5. Put an ext4 filesystem on the Linux partition.
6. Create a mount point called `/mnt/mypart` and mount the Linux partition on it.
7. Enable the swap partition and turn it on so that additional swap space is immediately available.
8. Create a volume group called `abc` from the LVM partition, create a 200MB logical volume from that group called `data`, add a VFAT partition, and then temporarily mount the logical volume on a new directory named `/mnt/test`. Check that it was successfully mounted.
9. Grow the logical volume from 200MB to 300MB.
10. Do what you need to do to remove the USB flash drive safely from the computer: unmount the Linux partition, turn off the swap partition, unmount the logical volume, and delete the volume group from the USB flash drive.